Cuando se trata de unidades de medida en CSS, se pueden distinguir dos grandes grupos, las **unidades absolutas** y las **unidades relativas**. En lo que se refiere al tamaño de fuente tipográfica, la unidad absoluta estrella es sin duda el píxel (`px`), y entre las unidades relativas destacan las unidades `em` y `rem`, junto al viejo porcentaje (`%`).

Las dimensiones de muchos elementos gráficos cambian junto al tamaño de la fuente, algo mucho **más fácil de mantener utilizando unidades relativas**, especialmente en diseño responsive.

Por ejemplo, si cambiamos el `font-size` del elemento superior en un determinado breakpoint, el tamaño de todos los elementos inferiores se actualizará automáticamente. Por el contrario, si se utilizan unidades absolutas, hay que actualizar el tamaño de todos los elementos en cada breakpoint.

En este contexto, hay quien opina que es mejor utilizar unidades `rem` y hay quien opina que es mejor utilizar unidades `em`, también quien no tiene problema en combinar ambas unidades. En este post trataremos de aprender que son exactamente las unidades `em` y las unidades `rem`, y como la combinación de ambas puede ser una opción interesante en algunas situaciones.

¿Que es em?

La unidad `em` es una unidad que nació en el mundo de la impresión y que es igual al tamaño de punto (`point-size`). Así, si un elemento tiene un `point-size` de 10, entonces `1em = 10point-size`.

En CSS no se utiliza el tamaño de punto y esta definición no es directamente aplicable, en su lugar se utiliza el **tamaño de fuente**, la propiedad `font-size`. En otras palabras, las unidades `em` son un **múltiplo del tamaño de fuente del elemento**.

Por ejemplo, si para un selector CSS hemos especificado un `font-size` de 16px, entonces `1em = 16px:`

```
p {
  font-size: 16px; /* 1em = 16px */
}
```

Y como CSS se aplica en cascada, la propiedad `font-size` puede ser heredada de los elementos superiores. Esto muy útil en diseño responsive, pues permite utilizar un valor de `font-size` para el elemento raíz `<html>` y utilizar `em` para el resto de elementos. De este modo, cambiando el `font-size` para `<html>` en un determinado breakpoint, **se actualizarán los valores de todo el documento**.

Generalmente, el font-size para `<html>` se establece en el 100%, lo que equivale al tamaño de fuente de la configuración del navegador, cuyo valor predeterminado suele ser de 16px:

```
html {
```

```
  font-size: 100%; /* 100% = 16px */
}
p {
  font-size: 1em; /* 1em = 16px */
}
h1 {
  font-size: 2em; /* 2em = 32px */
}
```

Algunos desarrolladores prefieren utilizar un `font-size` de 62.5% para `<html>`. De este modo reducen el valor predeterminado de 16px hasta 10px y así pueden realizar los cálculos en múltiplos de 10, más simple y sencillo; por ejemplo, 1em = 10px, 1.2em = 12px, etc. Esto ya es a gusto de cada uno, yo me quedo con 100%.

Imaginemos que queremos utilizar valores más altos para varias medidas, y de varios elementos, para una anchura de viewport superior a 1200px, y además queremos seguir una escala 1:1.3:

```
html {
  font-size: 100%; /* 100% = 16px */
}
h1 {
  font-size: 2.197em; /* 2.197 × 16px = 35px; obtenido de (1.3)³em */
}
h2 {
  font-size: 1.69em; /* 1.69em × 16px = 27px; obtenido de 2.197/1.3 */
}
h3 {
  font-size: 1.3em; /* 1.3em × 16 = 21px; obtenido de 1.69/1.3 */
}
small, .small {
  font-size: 0.77em /* 0.77em × 16 = 12.32px; obtenido de 1/1.3 */
}
@media (min-width: 1200px) {
  html {
    font-size: 1.2em; /* 19px */
  }
  /**
   * El font-size del resto de elementos
   * se actualizará automáticamente
   */
}
```

Utilizando `px`, habría que recalcular los píxeles de todos los elementos dentro del media query. Al utilizar `em`, nos basta con redeclarar el `font-size` en `<html>` y nada más.

A la hora de utilizar `em` es importante recordar que si se anidan elementos, el `font-size` de elementos superiores afectará al valor de `em` en elementos inferiores, y no solo el `font-size` del

elemento raíz.

Por ejemplo, si tenemos este HTML:

```
<div class="article">
  <p>Lorem fistrum quietooor nostrud exercitation pupita sed qui apetecan ese que llega jarl esse. Torpedo occaecat te va a hasé pupitaa eiusmod. Pecador va usté muy cargadoo quis quis minim.</p>
</div>
```

Y asignamos `2em` a `.article`, para que en `<p>` volvamos a tener 16px habría que tener este CSS:

```
html {
  font-size: 100%; /* 100% = 16px */
}
.article {
  font-size: 2em; /* Ahora 1em = 32px en elementos anidados */
}
p {
  /* Se hereda el font-size, así que aquí 1em = 32px */
  font-size: 0.5em; /* 0.5 x 32px = 16px */
}
```

em para padding y margin

Además de `font-size`, `em` se puede utilizar como unidad de medida en cualquier otra propiedad CSS. Como `em` es relativa al `font-size` del elemento, **será muy útil en propiedades que requieran mantener la proporcionalidad con el `font-size` del elemento**; por ejemplo, es muy habitual en `margin` y `padding`.

```
html {
  font-size: 100%; /* 100% = 16px */
}
h1 {
  font-size: 2.197em; /* 2.197 x 16px = 35px, así que para h1 1em = 2.197 */
  margin-top: 0.59em; /* 0.59 em = 20px, ya que 35 x 0.59 = 20  */
  margin-bottom: 0.59em;
}
```

Fíjate bien en el ejemplo anterior. Aquí es dónde empieza la **principal dificultad de entender como utilizar `em`**. Para `h1`, el valor de `font-size` se hereda desde `html`, que era de 16px, y se establece en 2.197em, que equivale a 35px.

En ese momento, el `font-size` del elemento sería de 35px, es decir, para el resto de propiedades

en h1, **1em sería igual a 35px**, ya que, si recordamos la definición de `em`, **1em es igual al tamaño de fuente definido para el elemento actual**.

En otras palabras, **las propiedades definidas en `em` cambian con del `font-size` del elemento**.

¿Qué es rem?

La unidad `rem` viene de **Root EM**, pero es muy importante que el nombre no confunda. `rem` no es igual al `em` del elemento root, es **igual al `font-size` del elemento root**, que en HTML sería el elemento `<html>`.

Si el `font-size` del `<html>` es 16px, **1rem sería igual a 16px en cualquier parte del documento**. Como siempre se refiere al elemento root, no se ve afectado por el `font-size` de elementos anidados.

Si tomamos el último ejemplo y lo pasamos a `rem`, para obtener un margin de 20px en el elemento h1, tendríamos que cambiarlo por esto:

```
html {
  font-size: 100%; /* 100% = 16px */
}
h1 {
  font-size: 2.197rem; /* 2.197 × 16px = 35px */
  margin-top: 1.25rem; /* 1.25rem = 20px, ya que 1.25 × 16 = 20  */
  margin-bottom: 1.25rem;
}
```

Debido a que 1rem es constante a lo largo del documento, **los cálculos a realizar son más sencillos que con em**.

Utilizando em y rem a la vez

Durante mucho tiempo se ha recomendado utilizar solo unidades `em`, ya que permite **diseñar componentes completamente modulares**, aunque los cálculos sean un poco más tediosos. Otros, aunque pierdan modularidad, prefieren utilizar solo unidades `rem`, pues todos los cálculos parten del mismo valor, el `font-size` del elemento raíz.

Pero si has prestado atención, entenderás la siguiente deducción: para un selector CSS dado, las propiedades definidas **en `em` cambian con el `font-size` del elemento actual**, las propiedades definidas **en `rem` siempre cambian con el `font-size` del elemento `<html>`**.

De ahí que cada vez haya más y más ejemplos de uso de ambas unidades a la vez, incluso dentro de la misma propiedad. La clave está en la deducción anterior:

1. **Utiliza `em` para propiedades que cambien con el `font-size` del elemento**

2. **Utiliza `rem` para todo lo demás**

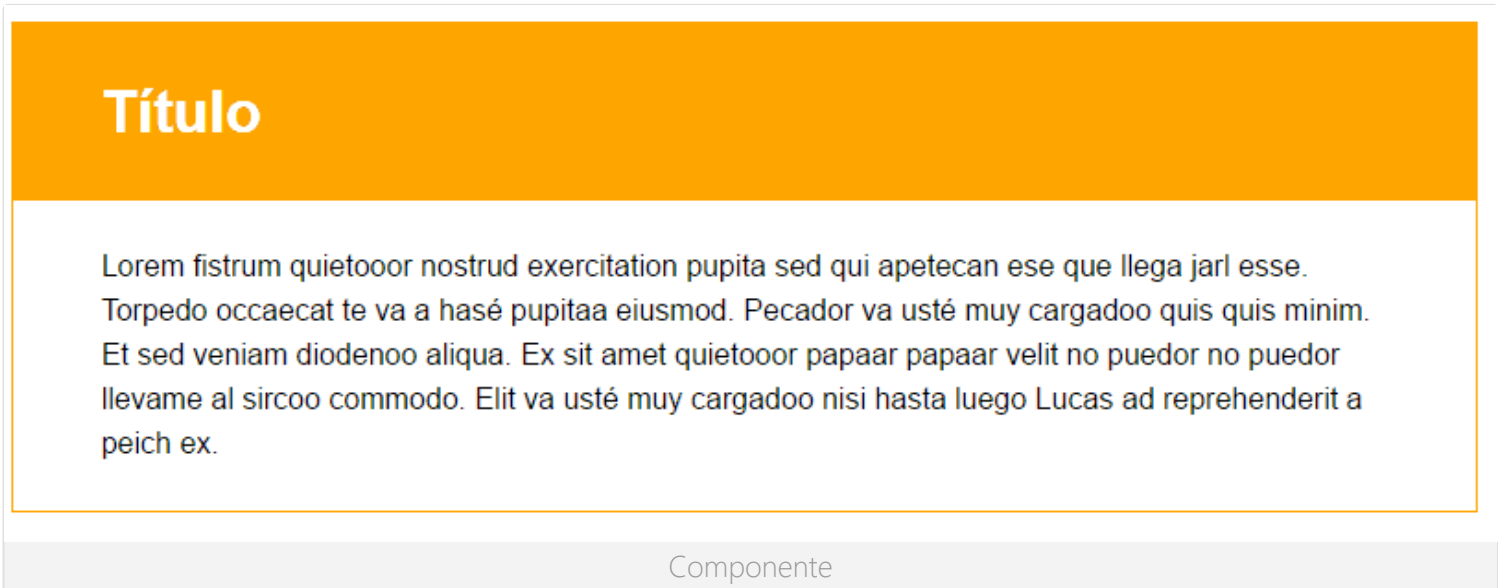
Ejemplo: alineando contenido de un componente

Un ejemplo en el que se entiende muy bien como se puede combinar el uso de `em` y `rem` es la alineación de contenido en un componente, como muy bien explicaba Zell en su artículo REM vs EM.

Vamos a hacer este componente:

```
<div class="component">
  <div class="component__header">Título</div>
  <div class="component__content">
    Lorem fistrum quietooor nostrud exercitation pupita sed qui apetecan ese que llega jarl esse. Torpedo occaecat te va a hasé pupitaa eiusmod. Pecador va usté muy cargadoo quis quis minim. Et sed veniam diodenoo aliqua. Ex sit amet quietooor papaar papaar velit no puedor no puedor llevame al sircoo commodo. Elit va usté muy cargadoo nisi hasta luego Lucas ad reprehenderit a peich ex.
  </div>
</div>
```

Que tendrá este aspecto final:



Componente

Se podría hacer todo con `em`:

```
.component {
  width: 100%;
  border: 1px orange solid;
}
.component__header {
```

```
  font-size: 2em;
  background-color: orange;
  color: white;
  padding: 0.75em 1.5em;
  margin: 0;
}
.component__content {
  font-size: 1em;
  padding: 1.5em 3rem;
  margin: 0;
}
```

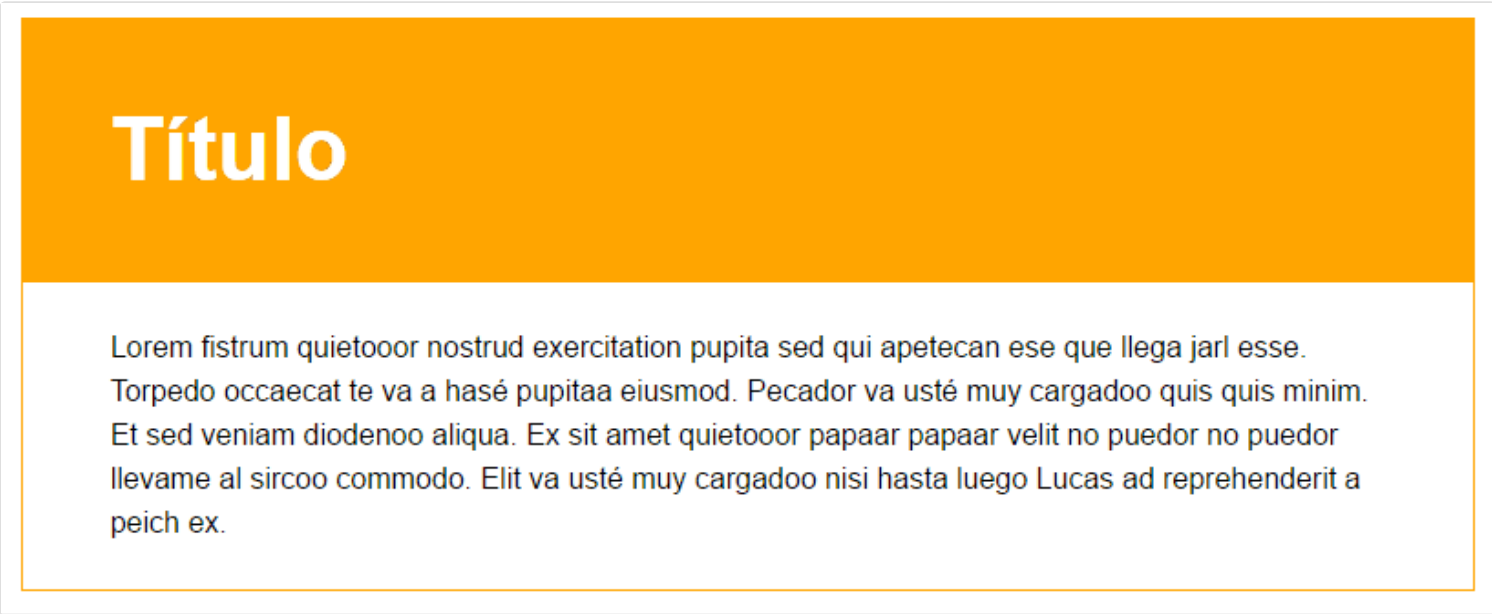
Fíjate como `.component__content` tiene la mitad de `font-size` que `.component__header`, por eso se le asigna el doble de padding y ambos elementos mantienen el mismo padding.

También se podría hacer todo con `rem`:

```
.component__header {
  font-size: 2rem;
  background-color: orange;
  color: white;
  padding: 1.5rem 3rem;
  margin: 0;
}
.component__content {
  font-size: 1rem;
  padding: 1.5rem 3rem;
  margin: 0;
}
```

Fíjate ahora que el padding es igual para ambos selectores, ya que `rem` siempre es el `font-size` del elemento root, no cambia con el `font-size` del elemento.

Ahora bien, hagamos una variante del componente con una cabecera más grande, con un `font-size` 1.5 veces la original, es decir, 3em o 3rem. El resultado perseguido es este:



Componente cabecera grande

Haciéndolo todo con em

Si cambiamos el tamaño de fuente de la cabecera a `3em`:

```
.component__header {
  font-size: 2em;
  background-color: orange;
  color: white;
  padding: 0.75em 1.5em;
  margin: 0;
}
.component__header--large {
  font-size: 3em;
}
.component__content {
  font-size: 1em;
  padding: 1.5em 3rem;
  margin: 0;
}
```

La cabecera se desplaza y no queda alineada con el contenido, el `padding-left` y el `padding-right` no son correctos:

Componente con cabecera grande desalineada

Esto ocurre porque, al definir el padding con `em`, el padding cambia al cambiar el `font-size` del elemento. Para solucionarlo y mantener el uso de `em`, habría que recalcular el padding left y right para `.component__header--large`:

```
.component__header--large {
  font-size: 3em; /* font-size aumenta 1.5 veces */
  padding-left: 1em; /* padding left y right disminuye 1.5 veces */
  padding-right: 1em;
}
```

Haciéndolo todo con rem

Se podría hacer todo con `rem` y así no tendríamos que redefinir el padding left y right:

```
.component__header {
  font-size: 2rem;
  background-color: orange;
  color: white;
  padding: 1.5rem 3rem;
  margin: 0;
```

```
}
.component__header--large {
  font-size: 3rem;
}
.component__content {
  font-size: 1rem;
  padding: 1.5rem 3rem;
  margin: 0;
}
```

Pero ahora el padding superior e inferior deja poco espacio a la cabecera, no mantiene las proporciones deseadas:

Padding top y bottom demasiado pequeño

Para solucionarlo, habría que redeclarar el padding top y bottom para `.component__header--large`:

```
.component__header--large {
  font-size: 3rem;
```

```
  padding-top: 2.16rem;
  padding-bottom: 2.16rem;
}
```

Haciéndolo con em y rem

Si te fijas detenidamente, hemos diseñado el elemento de forma que el `padding-top` y `padding-right` cambia con el `font-size` del elemento mientras que `padding-left` y `padding-right` no. Por eso había que redeclarar diversos paddings si se hacía todo en `em` o todo en `rem`.

Pero **si se mezclan ambas unidades no hay redeclarar nada**. Se siguen estos pasos:

1. Utilizamos unidades `rem` **para el** `font-size`; así, el `font-size` del componente o sus elementos no depende de elementos superiores, solo del root (útil si se sigue la metodología BEM).
2. Utilizamos unidades `em` **para las propiedades que cambien con el** `font-size` **del elemento**; en nuestro caso `padding-top` y `padding-bottom`.
3. Utilizamos `rem` **para todas las demás propiedades**; en nuestro caso `padding-left` y `padding-right`.

El CSS del componente quedaría así:

```
.component {
  width: 100%;
  border: 1px orange solid;
}
.component__header {
  font-size: 2rem;
  background-color: orange;
  color: white;
  padding: 1.5em 3rem; /* mezclando em y rem */
  margin: 0;
}
.component__header--large {
 font-size: 3rem;
}
.component__content {
 font-size: 1rem;
 padding: 1.5em 3rem;
 margin: 0;
}
```

Y si vemos las dos versiones del componente juntas, tendrían este HTML:

```
<div class="component">
```

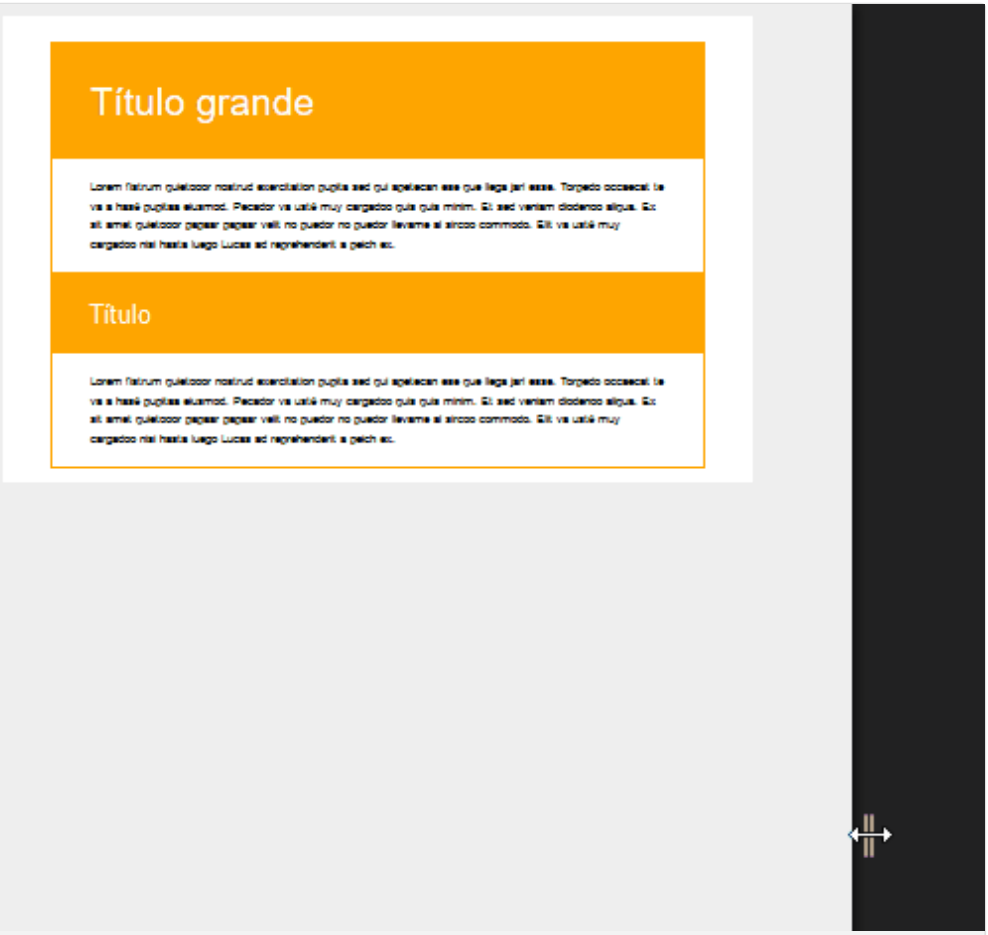
```
  <div class="component__header component__header--large">Título</div>
  <div class="component__content">
    Lorem fistrum quietooor nostrud exercitation pupita sed qui apetecan ese que llega jarl
esse. Torpedo occaecat te va a hasé pupitaa eiusmod. Pecador va usté muy cargadoo quis quis
minim. Et sed veniam diodenoo aliqua. Ex sit amet quietooor papaar papaar velit no puedor no
puedor llevame al sircoo commodo. Elit va usté muy cargadoo nisi hasta luego Lucas ad
reprehenderit a peich ex.
  </div>
</div>

<div class="component">
  <div class="component__header">Título</div>
  <div class="component__content">
    Lorem fistrum quietooor nostrud exercitation pupita sed qui apetecan ese que llega jarl
esse. Torpedo occaecat te va a hasé pupitaa eiusmod. Pecador va usté muy cargadoo quis quis
minim. Et sed veniam diodenoo aliqua. Ex sit amet quietooor papaar papaar velit no puedor no
puedor llevame al sircoo commodo. Elit va usté muy cargadoo nisi hasta luego Lucas ad
reprehenderit a peich ex.
  </div>
</div>
```

Y este aspecto:

Componente original y con cabecera grande

Y ahora ya, **el diseño responsive se podría ajustar automáticamente cambiando solo el `font-size` del root**:

```
@media (min-width: 1200px) {
  html {
    font-size: 125%;
  }
}
```



Resizing de componente con em y rem